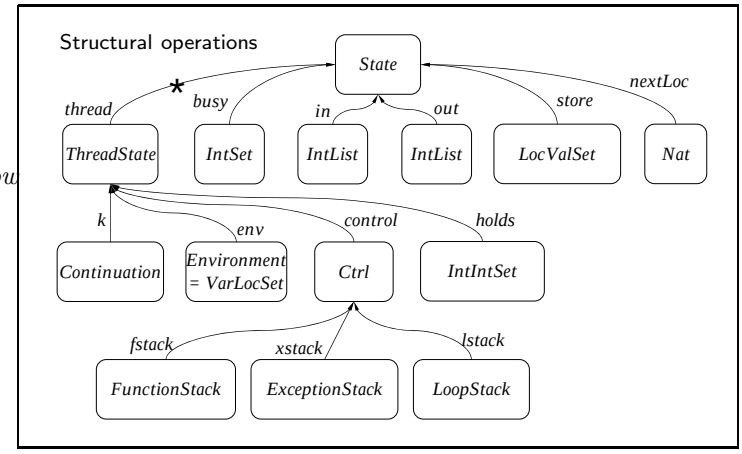
The definition of sequential FUN in Figure 6 needs to change as follows:
1) add callcc and its two rules below
2) replace structural operators with the ones in the picture to the right
3) replace eval and result definitions as below
4) declare non-deterministic the rules for write, variable lookup, read and print
5) add six more rules for threads:
a) one rule for creation of threads and one for termination of threads;
b) two rules for acquiring a lock
c) two rules for releasing a lock
No other changes needed.



$$newThread : Exp \times Environment \to StateAttribute\} \ \ldots\ \left\{ \frac{eval(E, Il)}{result(newThread(E,\cdot)\ busy(\cdot)\ in(Il)\ out(\cdot)\ store(\cdot)\ nextLoc(0))} \quad \frac{result(busy(\_)\ in(\_)\ out(Il)\ store(\_)\ nextLoc(\_))}{Il} \quad \frac{newThread(E, Env)}{thread(k(E)\ env(Env)\ control(fstack(\cdot)\ xstack(\cdot)\ lstack(\cdot))\ holds(\cdot))} \right.$$

$$\left.\begin{array}{l} \underline{\mathsf{callcc}} : Exp \to Exp\ [!] \\ cc : Continuation \times Ctrl \times Environment \to Val \end{array}\right\} \ldots\ \left\{ \frac{(k(\frac{V : Val \curvearrowright callcc}{(V, cc(K, C, Env)) \curvearrowright app} \curvearrowright K)\ C{:}Ctrl)\ env(Env)}{(k(\frac{((cc(K, C, Env), V) \curvearrowright app \curvearrowright \_)}{V \curvearrowright K}\ \frac{\_ : Ctrl}{C}\ env(\frac{\_}{Env})} \right.$$

$$\left.\begin{array}{l} \underline{\mathsf{spawn}} : Exp \to Exp \\ \underline{\mathsf{acquire}} : Exp \to Exp\ [!] \\ \underline{\mathsf{release}} : Exp \to Exp\ [!] \end{array}\right\} \ldots\ \left\{ \begin{array}{l} \frac{thread\langle k(\underline{\mathsf{spawn}}(E))\rangle\ env(Env)\rangle}{unit}\ \frac{\cdot}{newThread(E, Env)} \\ \frac{thread\langle k(\_ : Val)\ holds(LCs)\rangle}{\cdot}\ busy(\frac{Is}{Is - LCs}) \mid k(\frac{int(I) \curvearrowright acquire\rangle}{unit}\ holds\langle(I, \frac{N}{s(N)})\rangle \\ \boxed{k(\frac{int(I) \curvearrowright acquire\rangle}{unit}\ holds\langle\frac{\cdot}{(I,0)}\rangle\ busy(\frac{Is}{Is, I}) \Leftarrow not(I\ in\ Is)} \\ k(\frac{int(I) \curvearrowright release\rangle}{unit}\ holds\langle(I, \frac{s(N)}{N})\rangle \mid k(\frac{int(I) \curvearrowright release\rangle}{unit}\ holds\langle\frac{(I,0)}{\cdot}\rangle\ busy\langle\frac{I}{\cdot} \end{array} \right.$$

Figure 7: Adding call/cc and threads to FUN

as to throwing exceptions, exactly like in the original context. Therefore, in our language, call/cc appears to better abbreviate "call with current context".

Even though call/cc is conceptually more powerful than our other control-intensive language constructs (their translation would be technically involved, though), it is actually very easy to define in our framework. A special "current context" value is needed, so we define an operation $cc : Continuation \times Ctrl \times Environment \to Val$, as well as a corresponding continuation item, $callcc :\to ContinuationItem$. Note that callcc is declared strict in its argument. When a value (expected to be a function closure, namely the evaluated argument expression of callcc) is passed to it at the top of the continuation, a special "control context" value is created and passed to its argument function:

$$(k(\frac{V : Val \curvearrowright callcc}{(V, cc(K, C, Env)) \curvearrowright app} \curvearrowright K)\ C{:}Ctrl)\ env(Env)$$

If the special control context value is ever passed a value, then the original execution context is